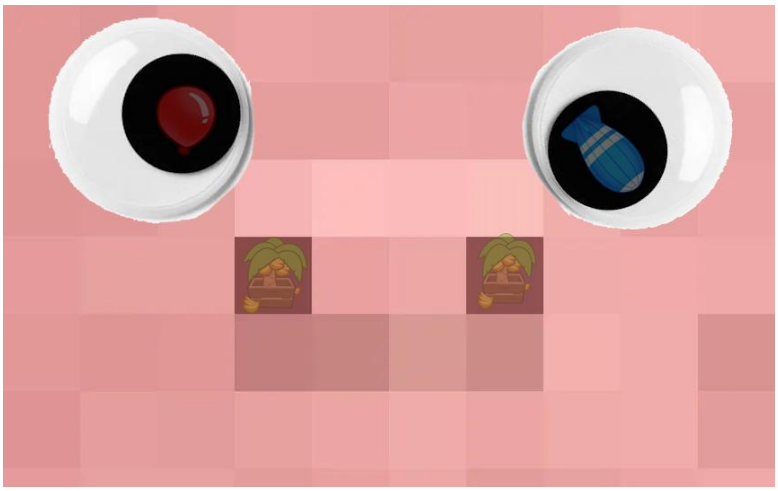

MineMobs Tower Defense

🔨 Block by Block 🔨

🎯 Pop Pop Pop 🎯

Name	Role	Cell Phone	Alternate Email
Ashley Ahn	Design	(404) 953-1062	Ashleyahn.30@gmail.com
Matthew Elledge	Programming	(404) 207-2916	Mtelledge14@gmail.com
AnnaGrace Gwee	Design	(478) 278-0198	Annagracegwee@gmail.com
Bryan Nguyen	Documentation	(470) 237-9923	Bryann0000@yahoo.com
Logan Slicker	Programming	(404) 983-0209	Lslicker24@gmail.com
Sharon Perry	Project Advisor	(470) 555-1212	Sperry46@kennesaw.edu
Norman Reid	Sponsor	(470) 578-5931	Nreid10@kennesaw.edu

Version 1.0 | [February 8th, 2025]

Table of Contents

Overview

The KSU Minecraft server hosts a multitude of game modes that we intend to add to: MineMobs Tower Defense! Our game mode provides a fun, co-opable genre that can be easily played with minimal effort. MineMobs Tower Defense is a stand-alone game mode that will parallel other market successes (Orcs Must Die, Bloons Tower Defense, Plants vs Zombies, Sanctum, etc.) with our own Minecraft spin to it. Anyone playing on the KSU MC Server can pick up and play the game mode!

Gameplay

- Objective: Defend your Village!
    - Players will have to hold against mobs looking to swarm their base/village
    - Survive a set number of waves (Campaign) / Survive as long as possible (Endless)
    - Players with the fastest clear time are given Leaderboard status (Time to think + time to clear mob waves)
- Game Progression: Players & Towers
    - Towers can be built and supplied with a villager to help auto defend
    - Players can interact with a GUI to build towers or manually defend with a sword/bow
    - As Mobs are “popped”, players will gain income to spend on progression
    - Progression is limited to the current map stage excluding potential “global” rouge-like buffs obtained outside of the current map
- Challenge: Difficulty Scaling
    - Less total lives
    - Reduced income
    - Increased Mob count per wave

Mechanics

- Rules
  - Players can only build towers with enough currency
  - Mobs are confined to walk the pre-determined map path
  - As Mobs are defeated, they lose a “tier” and drop down to a weaker mob
  - If a mob makes it to the end of the path, players lose a “life”
  - If the player loses all “lives”, the campaign/endless mode ends
- Game Structure
  - Players obtain $1 per Mob tier reduction or defeat
  - Players gain no money should a mob reach the end of the path
  - Players are shown the next wave’s mobs before they start it
- Character Actions
  - Players can interact with towers to place villagers into them
  - Players can interact with towers to upgrade them
  - Players can attack mobs on the path with a sword/bow
  - Players can start the next wave with a toggle switch
- Replaying and Saving
  - After a wave ends, the player’s wave progress is saved
  - If the wave has ended, a player can leave
  - If the wave has not ended, the last known save can be reloaded

Story & Narrative

While not necessary, it would be fun to have a background story or lore books for the game mode, being rewarded for completing certain achievements, maps, or restraints within time.

Game World

The game world will be reminiscent of BTD, with defined paths for the mobs and predetermined spots for players to build towers at. The levels are broken into different maps for the campaign that can be loaded into chunks designated for the game mode.

Rough Ideas for Maps:

Avatars

- Player Avatar
- Mobs
    - Pigs are Tier 1
        - Chickens are a special Tier 1 that can fly (Requires special towers/player to use their bow)
    - Sheep are Tier 2
    - Cows are Tier 3
    - Ghasts are MOAB-equivalent (BTD)
    - Ender Dragons are ZOMG-equivalent (BTD)
- Towers
    - Villagers of a profession determine their type of “weapon”
    - A villager’s profession level will be utilized to show the upgrade level of their tower

Levels

- Map 1: Tutorial
  - Learn how to spend currency (Towers/Upgrades)
  - Learn how to start a wave of mobs
  - Learn how to hit mobs
  - Learn how to hit special mobs
- Map 2:
  - Easy difficulty
  - Linear map design
  - 10 Waves
- Map 3:
  - Moderate difficulty
  - Multiple possible mob paths
  - 15 Waves
- Map 4:
  - Hard difficulty
  - Paths with multiple turns and intersection paths
  - 20 Waves

User Interface

- Chat Commands (Teleport to the Hub)
    - /MTD - General command for Commands, Help, and Information
    - /MTD Hub - Warp to Hub
    - /MTD Party - Finds Party Members for TD Campaigns/Endless

- GUI for Map Selection (Rough Map Designs Below)
    - Map Preview
    - Map Selection

- GUI for Towers
    - General Information UI
        - Stats
        - Building
        - Upgrading
- Mob Wave
    - Incoming Mobs
    - Wave Start

System Architecture

Classification

Subsystems:

- Main Class
  - Central entry point for the plugin lifecycle, runtime calls, and configuration management.
  - Responsible for managing communication across all other subsystems.
- Game Logic Layer
  - Contains the core mechanics of the tower defense game, such as round progression, mob spawning, tower placement, AI logic, and win/loss conditions.
- Player Interaction Layer
  - Facilitates communication between the system and the players, handling commands, GUI updates, and notifications.
- Persistence Layer
  - Manages data storage and retrieval for player saves, leaderboards, and external server communication.

Classes:

- **Main Class**
  - TowerDefensePlugin: The entry point of the plugin, responsible for lifecycle management and subsystem initialization.
- **Game Logic Layer**
  - RoundManager: Manages game rounds.
  - MobSpawner: Handles spawning of mobs.
  - TowerSpawner: Handles tower creation and placement.
  - HitDetection: Processes interactions between entities.
  - Win_LossDetection: Monitors and evaluates game state transitions.
  - TowerAI: Controls tower behaviors.
  - MobAI: Governs mob pathfinding and actions.
- **Player Interaction Layer**
  - CommandListeners: Manages player commands and triggers appropriate game logic.

    - GUIUpdater: Updates the graphical user interface based on game state changes.
- **Persistence Layer**
    - PlayerSaves: Manages saving and loading player data.
    - LeaderboardUpdater: Tracks and updates leaderboard data.
    - OtherServerData: Handles external server synchronization and data sharing.

Definition

Main:

- Serves as the entry point of the plugin. It initializes configurations, processes runtime calls, and manages the flow of information between the subsystems.

Game Logic:

- RoundManager
  - Controls the flow of the game by managing the start and end of rounds and scaling difficulty across rounds.
- MobSpawner
  - Handles the spawning of mobs at predefined or dynamic locations, ensuring the correct number and type of mobs are created per round.
- TowerSpawner
  - Manages the placement of towers by players, ensuring that towers are correctly initialized and interact with the game world.
- HitDetection
  - Detects collisions between projectiles and mobs, calculating and applying damage accordingly.
- Win_LossDetection
  - Evaluates game state conditions to determine whether the player has won or lost a round or the entire game.
- TowerAI
  - Simulates the behavior of towers, including targeting mobs, firing projectiles, and upgrading based on player input.
- MobAI
  - Governs mob movement and behavior, including pathfinding, attacking objectives, and responding to tower actions.

Player interaction:

- CommandListeners
  - Processes player commands to interact with the game, such as starting a round, upgrading towers, or viewing statistics.
- GUIupdater
  - Updates graphical user interfaces to reflect player status, game state, and upgrade options in real time.

Persistence:

- PlayerSaves
    - Manages saving and loading of player-specific data, such as tower placements, round progress, and in-game currency, ensuring continuity across sessions.
- LeaderboardUpdater
    - Tracks and updates player rankings or achievements based on performance, storing them for future reference or display.
- OtherServerData
    - Handles data exchange with external systems, such as servers or plugins, to enable cross-server functionality or shared leaderboards.

Constraints

Main class:

- **Timing**: Configuration files must be loaded during the plugin's initialization phase.
- **Storage**: Limited to JSON format for simplicity and compatibility with Minecraft server tools.
- **Constraints**:
    - Must include predefined settings.
    - Changes to configuration require a plugin reload to take effect.

Game Logic:

1. **RoundManager()**
    - **Timing**: Round progression depends on the successful completion of the current round.
    - **Constraints**:
        - Only one active round can exist at a time.
        - Rounds must escalate in difficulty based on a predefined scaling system.
2. **MobSpawner()**
    - **Timing**: Mobs must spawn at precise intervals, respecting tick timing.
    - **Constraints**:
        - Spawn rates and mob types must match the current round’s configuration.
        - The number of mobs spawned must not exceed server performance thresholds.
3. **TowerSpawner()**
    - **Constraints**:
        - Towers must be placed on valid blocks defined by the game.
        - Maximum tower count per player must adhere to server resource limits.
4. **HitDetection()**
    - **Constraints**:
        - Must use Minecraft's entity hitboxes for collision calculations.
        - Damage values must adhere to game balance constraints.
        - False positives must be minimized.

5. **Win_LossDetection()**
    - **Constraints**:
        - Game state must only transition to "win" or "loss" once all conditions are verified.
        - Loss detection must prioritize preventing infinite loops).
6. **TowerAI()**
    - **Constraints**:
        - Towers must prioritize the nearest mob within range.
        - Attack speed must respect predefined cooldown times.
7. **MobAI()**
    - **Constraints**:
        - Mob movement must use efficient pathfinding to prevent server lag.
        - Mobs must not deviate from the defined path unless obstructed.

Player Interaction:

- **CommandListeners()**
    - **Constraints**:
        - Commands must verify player permissions before execution.
        - Invalid input must be handled gracefully.
        - Command execution must not block the main thread.
- **GUIUpdater()**
        - **Constraints**:
            - GUI updates must synchronize with game state.
            - The GUI must respect Minecraft's display constraints.

Persistence Layer:

1. **PlayerSaves()**
    - **Storage**: Must store data in a lightweight, structured format.
    - **Constraints**:
        - Player data must be saved asynchronously to avoid blocking the server.
        - Save operations must ensure data integrity in case of server crashes.
2. **LeaderboardUpdater()**
    - **Timing**: Updates must occur after the end of a game or significant events.
    - **Constraints**:

- Leaderboard updates must not overwrite existing valid data.
- Synchronization with external leaderboard systems must handle potential latency.

3. **OtherServerData()**
   - **Constraints**:
     - Data synchronization must be non-blocking and performed in the background.
     - Network failures or desynchronization must not disrupt gameplay.

Resources

Memory:

- **Description**:
  - Managed for storing active mobs, towers, player data, and game state.
  - Each mob, tower, or player action creates temporary objects that require efficient memory usage to avoid performance degradation.
- **Potential Issues**:
  - Excessive memory usage could occur during large-scale mob spawns or extensive data storage.
- **Resolution**:
  - Optimize data structures to store only essential information.
  - Use Java's garbage collector effectively by ensuring all unused references are cleared.

Async Processes:

- **Description**:
  - Used heavily for game logic tasks like:
    - Mob AI pathfinding.
    - Tower AI targeting and firing.
    - GUI updates and player commands.
  - Computationally intensive processes run asynchronously to reduce server lag.
- **Potential Issues**:
  - High CPU usage during mob pathfinding or simultaneous tower actions.
- **Resolution**:
  - Utilize Paper API's asynchronous tasks (BukkitRunnable and runTaskLaterAsync) to offload heavy computations.
  - Limit the number of mobs or towers active at any one time.

Storage:

- **Description**:
  - Persistent storage for:
    - PlayerSaves for progress, currency, upgrades.
    - LeaderboardUpdater for rankings, achievements.
  - Storage is handled via JSON to maintain a lightweight footprint.
- **Potential Issues**:
  - Data corruption or inconsistency due to concurrent writes.

- **Resolution**:
    - Synchronize write operations to storage to prevent race conditions.
    - Use backups for critical data like player saves.

Development

- Our GitHub Source Code
- IntelliJ Community
- IntelliJ Minecraft Development Plugin
- Spigot Development Guide
- Paper Development Guide